



Lecture 14 of 42

First-Order Logic: Unification, Inference Discussion: PS3, Constraint Logic

Monday, 24 September 2007

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/v9v3>

Course web site: <http://www.kddresearch.org/Courses/Fall-2007/CIS730>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Reading for Next Class:

Section 9.2 – 9.4, p. 275 – 295, Russell & Norvig 2nd edition



Lecture Outline

- Reading for Next Class: Section 9.2 – 9.4, R&N 2e
- Recommended : Nilsson and Genesereth (Chapter 5 online)
- Today
 - * Generalized Modus Ponens
 - * Unification
 - * Constraint logic
- Wednesday
 - * Resolution theorem proving
 - * Prolog as related to resolution
 - * MP4 & 5 preparations
- Friday
 - * Logic programming in real life
 - * Industrial-strength Prolog
 - * Lead-in to MP4
- Week of 01 Oct 2007: KR and Ontologies





Logical Agents: Review

Logical agents apply inference to a knowledge base to derive new information and make decisions

Basic concepts of logic:

- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundness: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Propositional logic suffices for some of these tasks

Adapted from slides by
S. Russell, UC Berkeley



Example Proof: Review

| | | |
|---|-----------------------|---|
| ● | Bob is a buffalo | 1. $Buffalo(Bob)$ |
| | Pat is a pig | 2. $Pig(Pat)$ |
| | Buffaloes outrun pigs | 3. $\forall x, y \text{ Buffalo}(x) \wedge Pig(y) \Rightarrow Faster(x, y)$ |
| | Bob outruns Pat | |

- Apply Sequent Rules to Generate New Assertions

| | |
|--------------------------|--|
| AI 1 & 2 | 4. $Buffalo(Bob) \wedge Pig(Pat)$ |
| UE 3, $\{x/Bob, y/Pat\}$ | 5. $Buffalo(Bob) \wedge Pig(Pat) \Rightarrow Faster(Bob, Pat)$ |
| MP 6 & 7 | 6. $Faster(Bob, Pat)$ |

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta}$$

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta}$$

$$\frac{\forall x \alpha}{\alpha\{x/\tau\}}$$

- Modus Ponens And Introduction Universal Elimination

Adapted from slides by
S. Russell, UC Berkeley





Knowledge Engineering

- **KE: Process of**
 - * Choosing logical language (basis of KR)
 - * Building KB
 - * Implementing proof theory
 - * Inferring new facts
- **Analogy: Programming Languages / Software Engineering**
 - * Choosing programming language (basis of software engineering)
 - * Writing program
 - * Choosing / writing compiler
 - * Running program
- **Example Domains**
 - * Electronic circuits (Section 8.3 R&N)
 - * Exercise
 - ⇒ Look up, read about [protocol analysis](#)
 - ⇒ Find example and think about KE process for your project domain



Unification: Definitions and Idea Sketch

A substitution σ unifies atomic sentences p and q if $p\sigma = q\sigma$

| p | q | σ |
|------------------|-----------------------|------------------------------|
| $Knows(John, x)$ | $Knows(John, Jane)$ | $\{x/Jane\}$ |
| $Knows(John, x)$ | $Knows(y, OJ)$ | $\{x/John, y/OJ\}$ |
| $Knows(John, x)$ | $Knows(y, Mother(y))$ | $\{y/John, x/Mother(John)\}$ |

Idea: Unify rule premises with known facts, apply unifier to conclusion

E.g., if we know q and $Knows(John, x) \Rightarrow Likes(John, x)$
 then we conclude $Likes(John, Jane)$
 $Likes(John, OJ)$
 $Likes(John, Mother(John))$

Adapted from slides by
S. Russell, UC Berkeley





Generalized Modus Ponens

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{q\sigma} \quad \text{where } p_i'\sigma = p_i\sigma \text{ for all } i$$

E.g. $p_1' = \text{Faster}(\text{Bob}, \text{Fat})$
 $p_2' = \text{Faster}(\text{Pat}, \text{Steve})$
 $p_1 \wedge p_2 \Rightarrow q = \text{Faster}(x, y) \wedge \text{Faster}(y, z) \Rightarrow \text{Faster}(x, z)$
 $\sigma = \{x/\text{Bob}, y/\text{Pat}, z/\text{Steve}\}$
 $q\sigma = \text{Faster}(\text{Bob}, \text{Steve})$

GMP used with KB of definite clauses (*exactly one positive literal*):
 either a single atomic sentence or
 (conjunction of atomic sentences) \Rightarrow (atomic sentence)
 All variables assumed universally quantified

Adapted from slides by
 S. Russell, UC Berkeley



Soundness of GMP

Need to show that

$$p_1', \dots, p_n', (p_1 \wedge \dots \wedge p_n \Rightarrow q) \models q\sigma$$

provided that $p_i'\sigma = p_i\sigma$ for all i

Lemma: For any definite clause p , we have $p \models p\sigma$ by UE

1. $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \models (p_1 \wedge \dots \wedge p_n \Rightarrow q)\sigma = (p_1\sigma \wedge \dots \wedge p_n\sigma \Rightarrow q\sigma)$
2. $p_1', \dots, p_n' \models p_1' \wedge \dots \wedge p_n' \models p_1'\sigma \wedge \dots \wedge p_n'\sigma$
3. From 1 and 2, $q\sigma$ follows by simple MP

Adapted from slides by
 S. Russell, UC Berkeley





Forward Chaining

When a new fact p is added to the KB
 for each rule such that p unifies with a premise
 if the other premises are known
 then add the conclusion to the KB and continue chaining

Forward chaining is data-driven
 e.g., inferring properties and categories from percepts

Adapted from slides by
 S. Russell, UC Berkeley



Example: Forward Chaining

Add facts 1, 2, 3, 4, 5, 7 in turn.
 Number in \square = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ $\square_{1a, \times}$
5. $Pig(Pat)$ $\square_{1b, \surd}$ \rightarrow 6. $Faster(Bob, Pat)$ $\square_{3a, \times}$, $\square_{3b, \times}$
 $\square_{2a, \times}$
7. $Slug(Steve)$ $\square_{2b, \surd}$
 \rightarrow 8. $Faster(Pat, Steve)$ $\square_{3a, \times}$, $\square_{3b, \surd}$
 \rightarrow 9. $Faster(Bob, Steve)$ $\square_{3a, \times}$, $\square_{3b, \times}$

Adapted from slides by
 S. Russell, UC Berkeley





Backward Chaining

When a query q is asked
 if a matching fact q' is known, return the unifier
 for each rule whose consequent q' matches q
 attempt to prove each premise of the rule by backward chaining

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

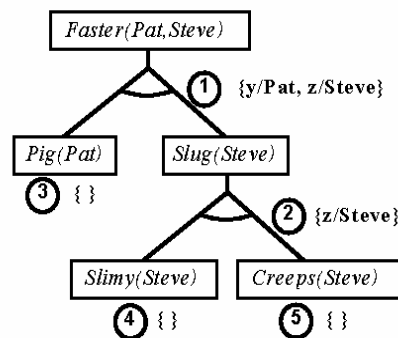
Backward chaining is the basis for logic programming, e.g., Prolog

Adapted from slides by
 S. Russell, UC Berkeley



Example: Backward Chaining

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$ 4. $Slimy(Steve)$ 5. $Creeps(Steve)$



Adapted from slides by
 S. Russell, UC Berkeley





Backward Chaining

When a query q is asked
 if a matching fact q' is known, return the unifier
 for each rule whose consequent q' matches q
 attempt to prove each premise of the rule by backward chaining

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

Backward chaining is the basis for logic programming, e.g., Prolog

- Answer
 - * Suppose ¬Query, For The Sake Of Contradiction (FTSOC)
 - * Attempt to prove that $KB \wedge \neg Query \vdash \perp$



Resolution Inference Rule

Basic propositional version:

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Full first-order version:

$$\frac{p_1 \vee \dots \vee p_j \dots \vee p_m, \quad q_1 \vee \dots \vee q_k \dots \vee q_n}{(p_1 \vee \dots \vee p_{j-1} \vee p_{j+1} \dots \vee p_m \vee q_1 \dots \vee q_{k-1} \vee q_{k+1} \dots \vee q_n)\sigma}$$

where $p_j\sigma = \neg q_k\sigma$

For example,

$$\frac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Me)}{Unhappy(Me)}$$

with $\sigma = \{x/Me\}$

Adapted from slides by
 S. Russell, UC Berkeley





Conjunctive Normal (aka Clausal) Form: Conversion (Nilsson) and Mnemonic

- Implications Out
- Negations Out
- Standardize Variables Apart
- Existentials Out (Skolemize)
- Universals Made Implicit
- Distribute *And* Over *Or* (i.e., Disjunctions In)
- Operators Out
- Rename Variables
- A Memonic for *Star Trek: The Next Generation* Fans

Captain Picard:

I'll Notify Spock's Eminent Underground Dissidents On Romulus

I'll Notify Sarek's Eminent Underground Descendant On Romulus

Adapted from slides by S. Russell, UC Berkeley



Skolemization

$\exists x \text{Rich}(x)$ becomes $\text{Rich}(G1)$ where $G1$ is a new "Skolem constant"

$\exists k \frac{d}{dy}(k^y) = k^y$ becomes $\frac{d}{dy}(e^y) = e^y$

More tricky when \exists is inside \forall

E.g., "Everyone has a heart"

$\forall x \text{Person}(x) \Rightarrow \exists y \text{Heart}(y) \wedge \text{Has}(x, y)$

Incorrect:

$\forall x \text{Person}(x) \Rightarrow \text{Heart}(H1) \wedge \text{Has}(x, H1)$

Correct:

$\forall x \text{Person}(x) \Rightarrow \text{Heart}(H(x)) \wedge \text{Has}(x, H(x))$

where H is a new symbol ("Skolem function")

Skolem function arguments: all enclosing universally quantified variables

Adapted from slides by S. Russell, UC Berkeley





Resolution Theorem Proving

To prove α :

- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

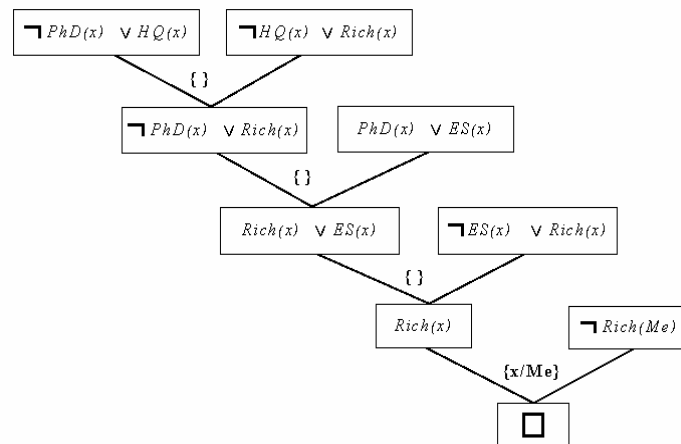
E.g., to prove $Rich(me)$, add $\neg Rich(me)$ to the CNF KB

- $\neg PhD(x) \vee HighlyQualified(x)$
- $PhD(x) \vee EarlyEarnings(x)$
- $\neg HighlyQualified(x) \vee Rich(x)$
- $\neg EarlyEarnings(x) \vee Rich(x)$

Adapted from slides by S. Russell, UC Berkeley



Example: Resolution Proof



Adapted from slides by S. Russell, UC Berkeley



Offline Exercise: Read-and-Explain Pairs

- For Class Participation (PS5)
 - With Your Assigned Partner(s)
-
- Read: Chapter 10 R&N
 - By 13 Oct 2007



Logic Programming vs. Imperative Programming

Sound bite: computation as inference on logical KBs

| <u>Logic programming</u> | <u>Ordinary programming</u> |
|-------------------------------------|---------------------------------|
| 1. Identify problem | Identify problem |
| 2. Assemble information | Assemble information |
| 3. Tea break | Figure out solution |
| 4. Encode information in KB | Program solution |
| 5. Encode problem instance as facts | Encode problem instance as data |
| 6. Ask queries | Apply program to data |
| 7. Find false facts | Debug procedural errors |

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2 !$





A Look Ahead: Logic Programming as Horn Clause Resolution

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques \Rightarrow 10 million LIPS

Program = set of clauses = head :- literal₁, ... literal_n.

Efficient unification by open coding

Efficient retrieval of matching clauses by direct linking

Depth-first, left-to-right backward chaining

Built-in predicates for arithmetic etc., e.g., X is Y*Z+3

Closed-world assumption ("negation as failure")

e.g., not PhD(X) succeeds if PhD(X) fails

Adapted from slides by S. Russell, UC Berkeley

CIS 530 / 730: Artificial Intelligence

Monday, 24 Sep 2007

Computing & Information Sciences
Kansas State University



A Look Ahead: Logic Programming (Prolog) Examples

Depth-first search from a start state X:

```
dfs(X) :- goal(X).
```

```
dfs(X) :- successor(X,S),dfs(S).
```

No need to loop over S: successor succeeds for each

Appending two lists to produce a third:

```
append([],Y,Y).
```

```
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

```
query: append(A,B,[1,2]) ?
```

```
answers: A=[] B=[1,2]
```

```
A=[1] B=[2]
```

```
A=[1,2] B=[]
```

Adapted from slides by S. Russell, UC Berkeley

CIS 530 / 730: Artificial Intelligence

Monday, 24 Sep 2007

Computing & Information Sciences
Kansas State University





Summary Points

- **From Propositional to First-Order Proofs**
 - * Generalized Modus Ponens
 - * Resolution
- **Unification Problem**
- **Roles in Computer Science**
 - * Type inference
 - * Theorem proving
 - * What do these have to do with each other?
- **Search Patterns**
 - * Forward chaining
 - * Backward chaining
 - * Fan-in, fan-out



Terminology

- **From Propositional to First-Order Proofs**
 - * Generalized Modus Ponens
 - * Resolution
- **Unification Problem**
- **Most General Unifier (MGU)**
- **Roles in Computer Science**
 - * Type inference
 - * Theorem proving
 - * What do these have to do with each other?
- **Search Patterns**
 - * Forward chaining
 - * Backward chaining
 - * Fan-in, fan-out

